Analytic Performance Modeling of Combustion Codes with ExaSAT

June 2013

Cy Chan, Didem Unat,
Gilbert Hendry, Mike Lijewski, Sam Williams,
Weiqun Zhang, John Bell, and John Shalf

ExaCT Combustion Codesign Center
CAL (Computer Architecture Lab)
LBL/Sandia/UCB

ExaCT: Combustion Co-Design

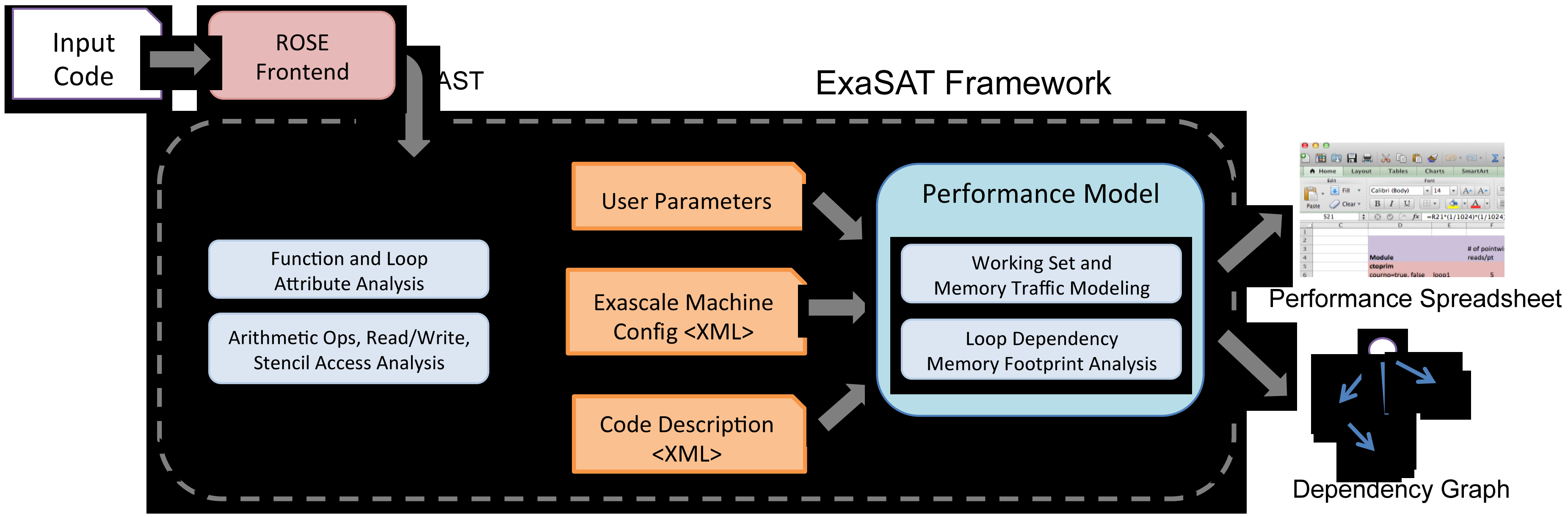
- Exascale Center for Combustion in Turbulence (ExaCT) is one of three exascale co-design centers
- Combustion accounts for 85% of the energy used in the US
  - Highly efficient combustion systems will help us meet the 80% reduction target of greenhouse gas emissions by 2050
- SMC is a proxy app for the S3D combustion simulation
  - 8th order finite difference code
  - Simulates chemistry interactions: 50+ species is the exascale target

Motivation for Analytic Model

- Answer co-design questions acquired from Fast Forward vendors
- Hardware implications
  - Assess baseline hardware requirements of combustion simulations
  - Make preliminary recommendations on architectural design choices and give feedback to vendors
- Software implications
  - Quickly explore software optimizations and their interaction with hardware trade-offs
  - Guide development of advance programming models and runtimes for combustion codes

ExaSAT: Exascale Static Analysis Tool

- Can automatically predict performance for many input codes and software optimizations
- Predict performance under different architectural scenarios
- Much faster than hardware simulation and manual modeling


Input Code
ROSE Frontend
AST
ExaSAT Framework
Function and Loop Attribute Analysis
Arithmetic Ops, Read/Write, Stencil Access Analysis
User Parameters
Exascale Machine Config <XML>
Code Description <XML>
Performance Model
Working Set and Memory Traffic Modeling
Loop Dependency Memory Footprint Analysis
Performance Spreadsheet
Dependency Graph

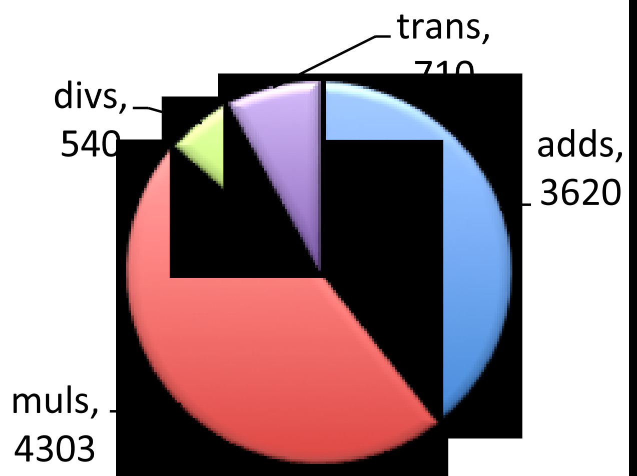
Performance Metrics

- The list of metrics that we used for evaluating various hardware components and software optimizations

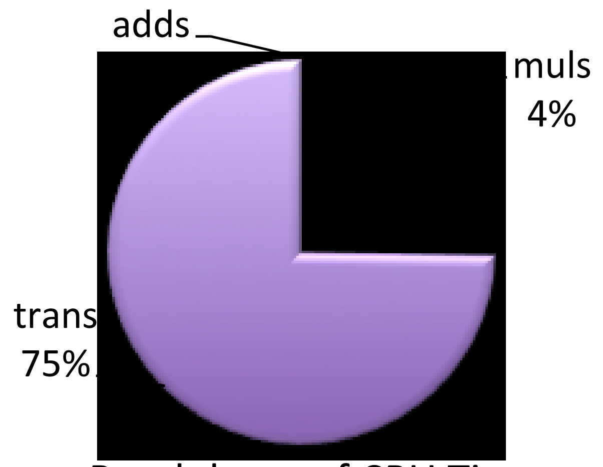
Metric	Corresponding Analysis
Memory Traffic & B:F Ratio	Sensitivity to the memory bandwidth as a result of data movement optimizations
Working Set Size	Data reuse strategies for filtering memory bandwidth
State Variables	Effect of number of registers to avoid register spilling
Arithmetic Operations	FP instruction mix, special hardware, & benefits of vectorization
Read/Write Ratio & Write Access Rate	Candidate streaming data for secondary nonvolatile memory
Fraction of Communication	On-node vs off-node data movement

SMC code
with 53 species

Chemistry

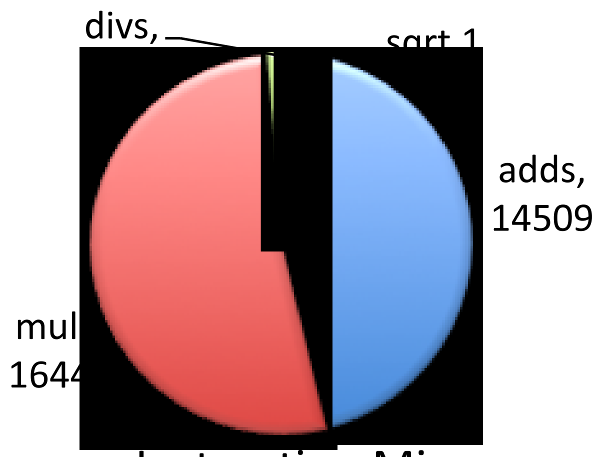


Instruction Mix

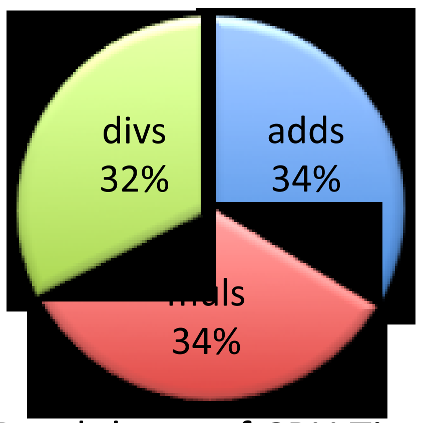


Breakdown of CPU Time

Dynamics



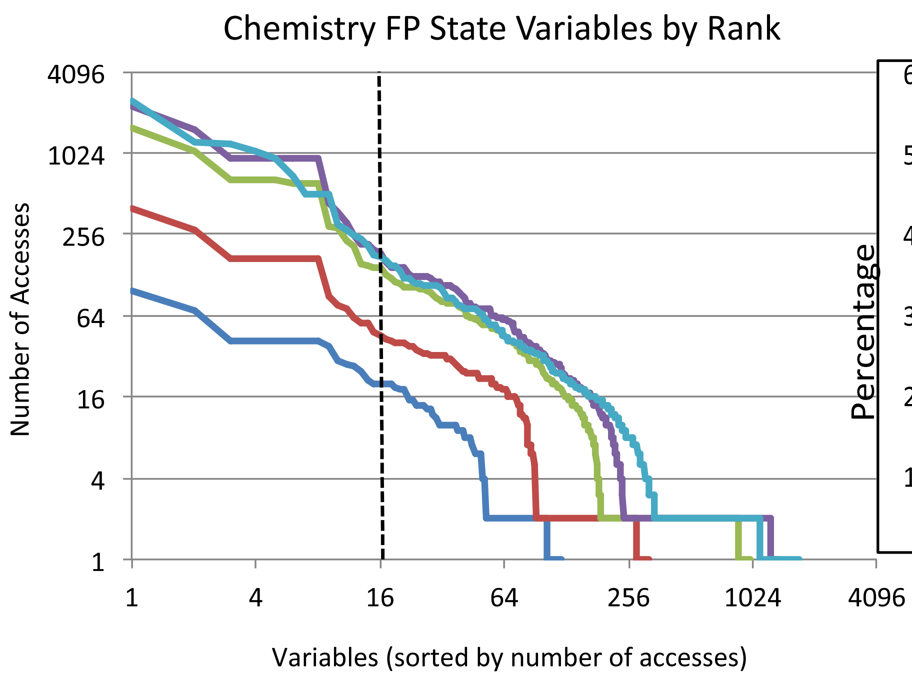
Instruction Mix

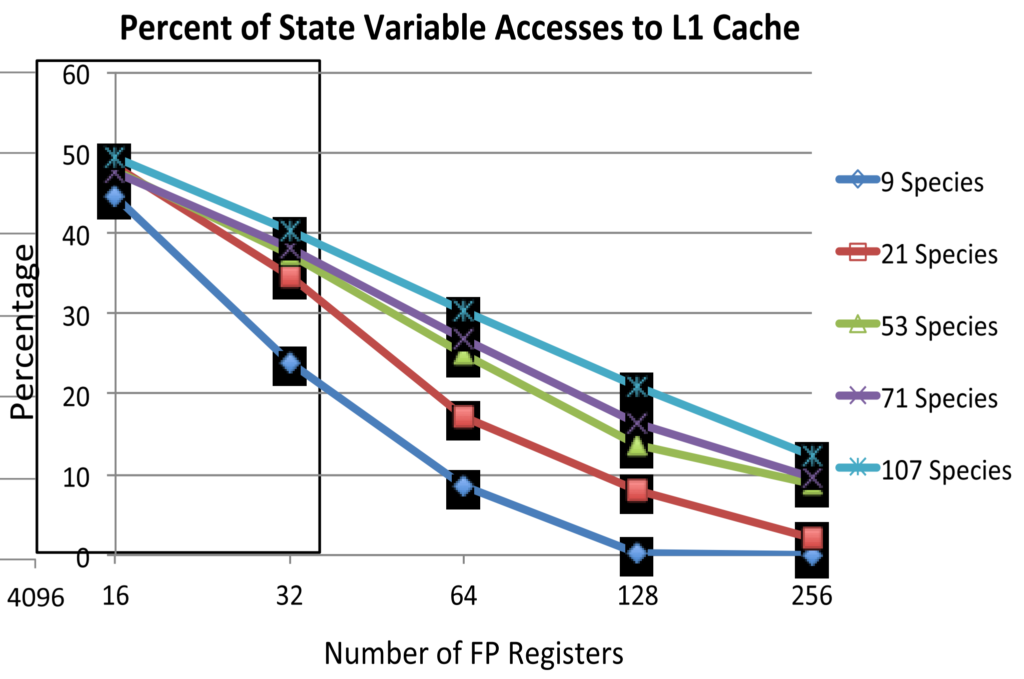


Breakdown of CPU Time

Even though transcendentals and division ops might be low in count, they can dominate the CPU time

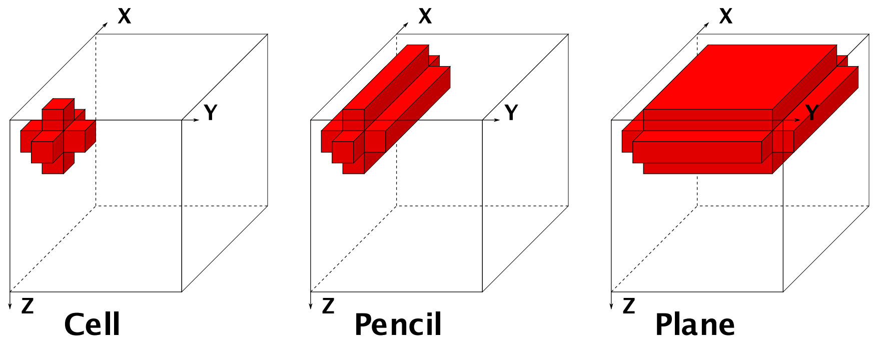
Registers and L1 Cache Traffic

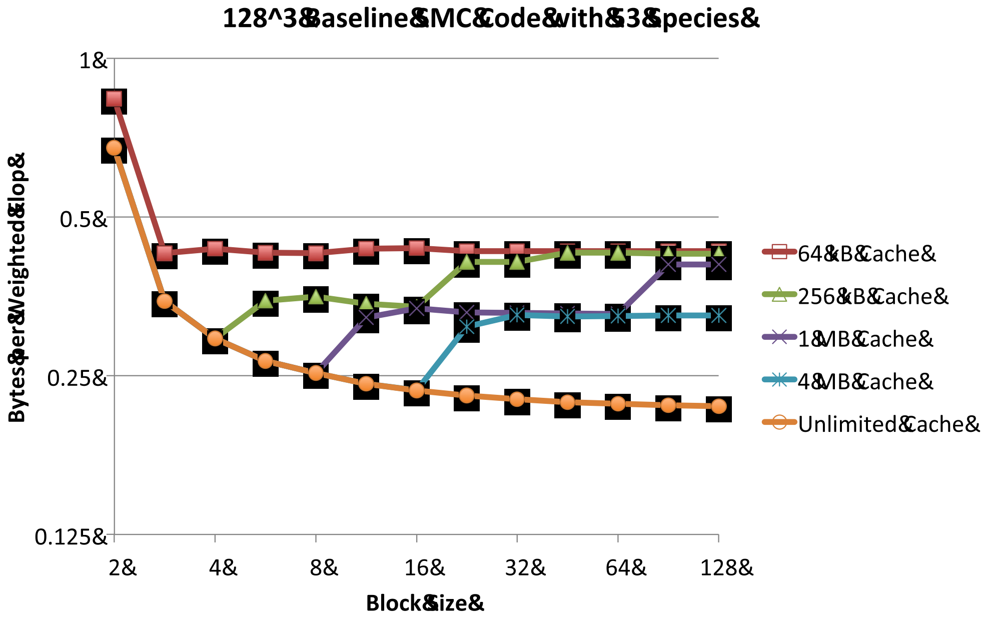




- Accesses to state variables that do not reside in a register result in additional L1 cache traffic
- Most (>95%) of the L1 cache traffic in chemistry code is from state variable accesses, and not the streaming data variables

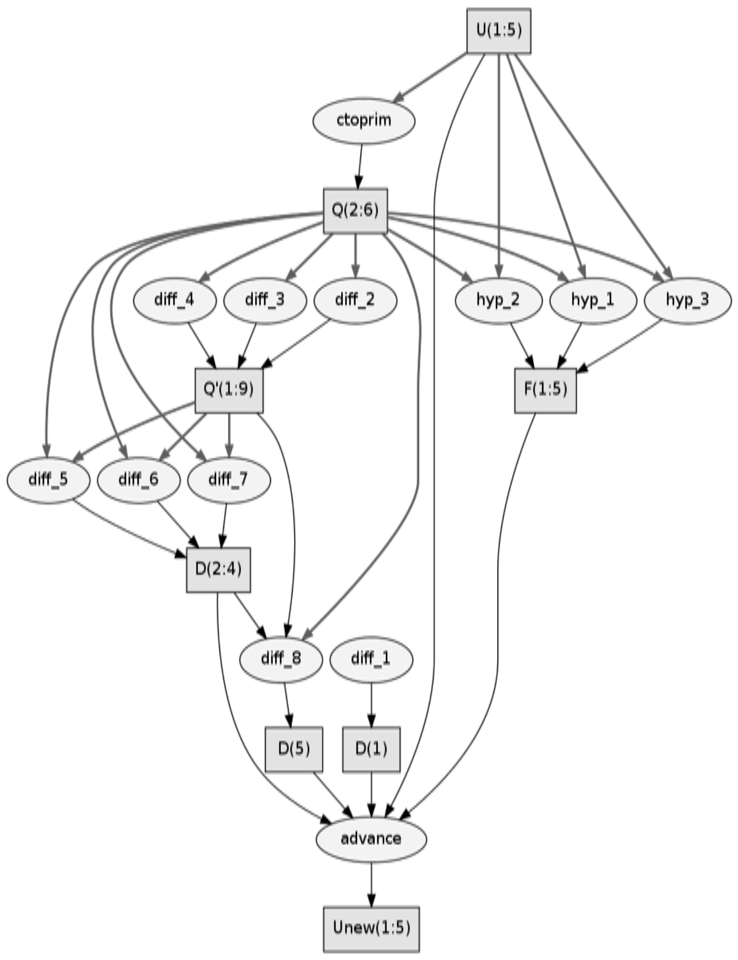
Cache Model





- Models fully-associative cache with LRU replacement policy
- Identifies data reuse for stencil computations based on working set and cache sizes
- Ideal model: determines the performance ceiling and identifies trade-offs in memory subsystem

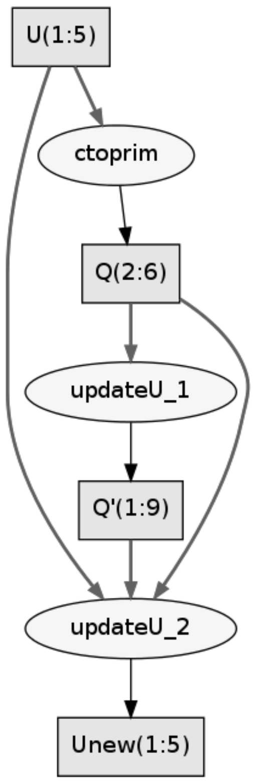
Loop Fusion Dependency Graph for CNS code



Baseline

2.9 GB/sweep

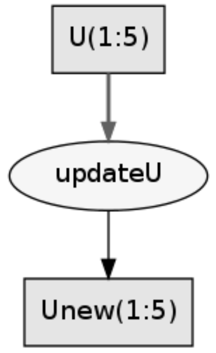
1.78 Bytes/Flop



Simple Fusion

1.6 GB/sweep (–46%)

0.96 Bytes/Flop

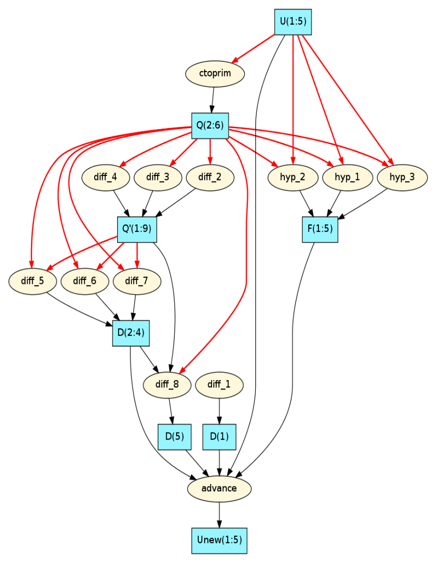


Aggressive Fusion

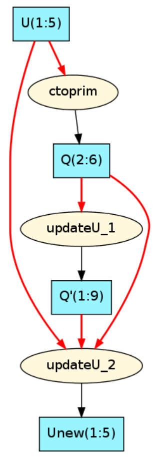
0.48 GB/sweep (–84%)

0.29 Bytes/Flop

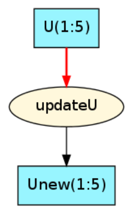
Impact of Software Optimization on CNS and SMC Dynamics



Baseline
2.9 GB/sweep
1.78 Bytes/Flop



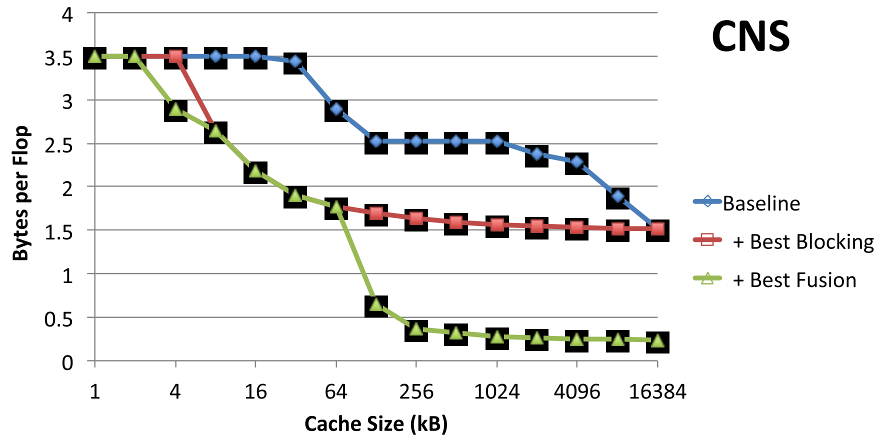
Simple Fusion
1.6 GB/sweep (–46%)
0.96 Bytes/Flop

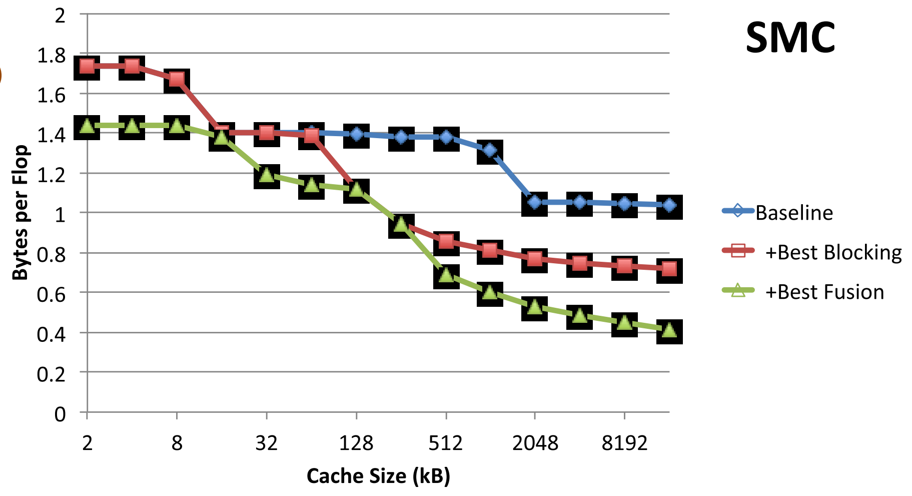


Aggressive Fusion
0.48 GB/sweep (–84%)
0.29 Bytes/Flop

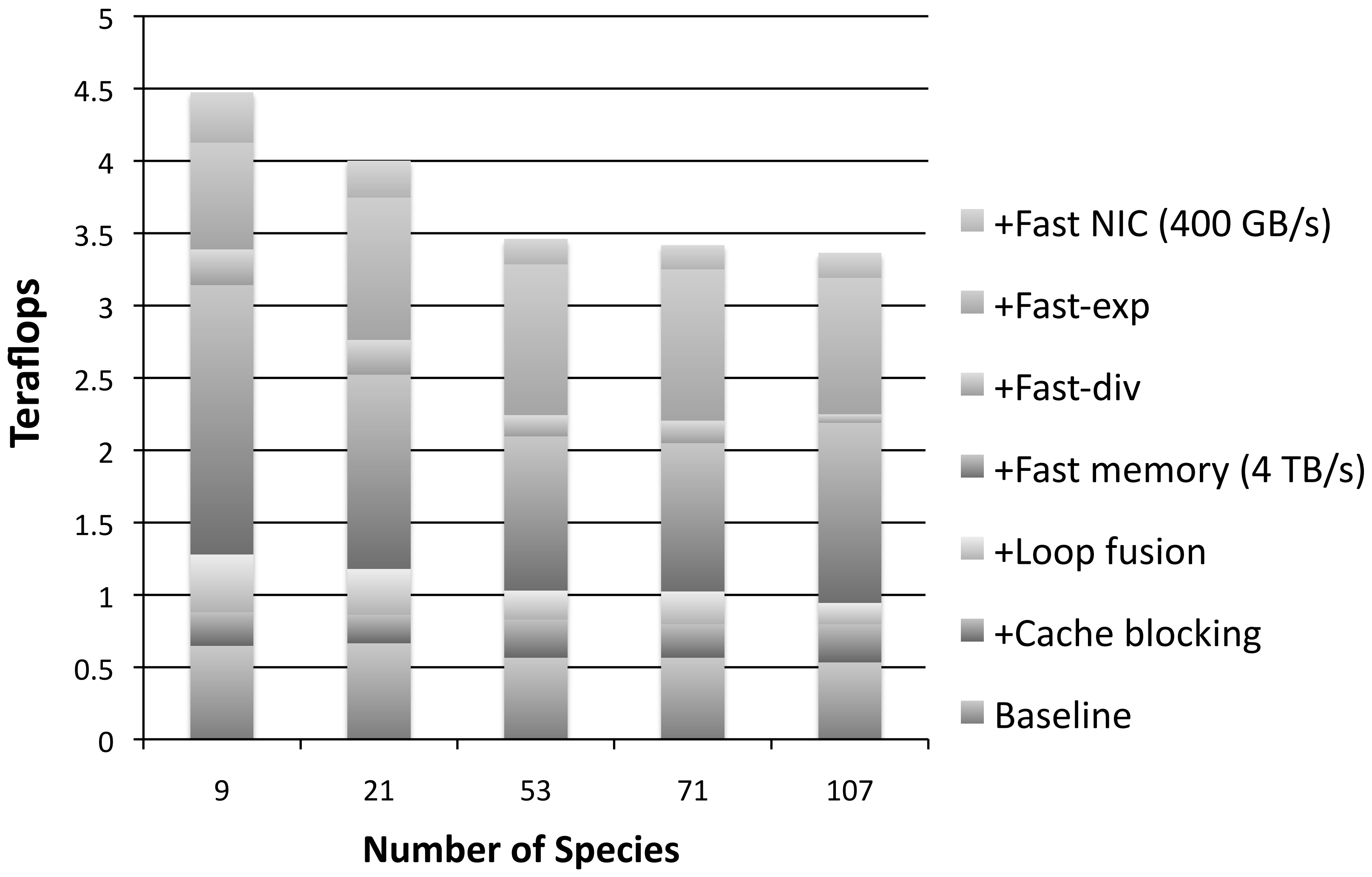
CNS Code Fusion Optimization

Software Optimization Impact on Byte to Flop Ratio vs. Cache Size



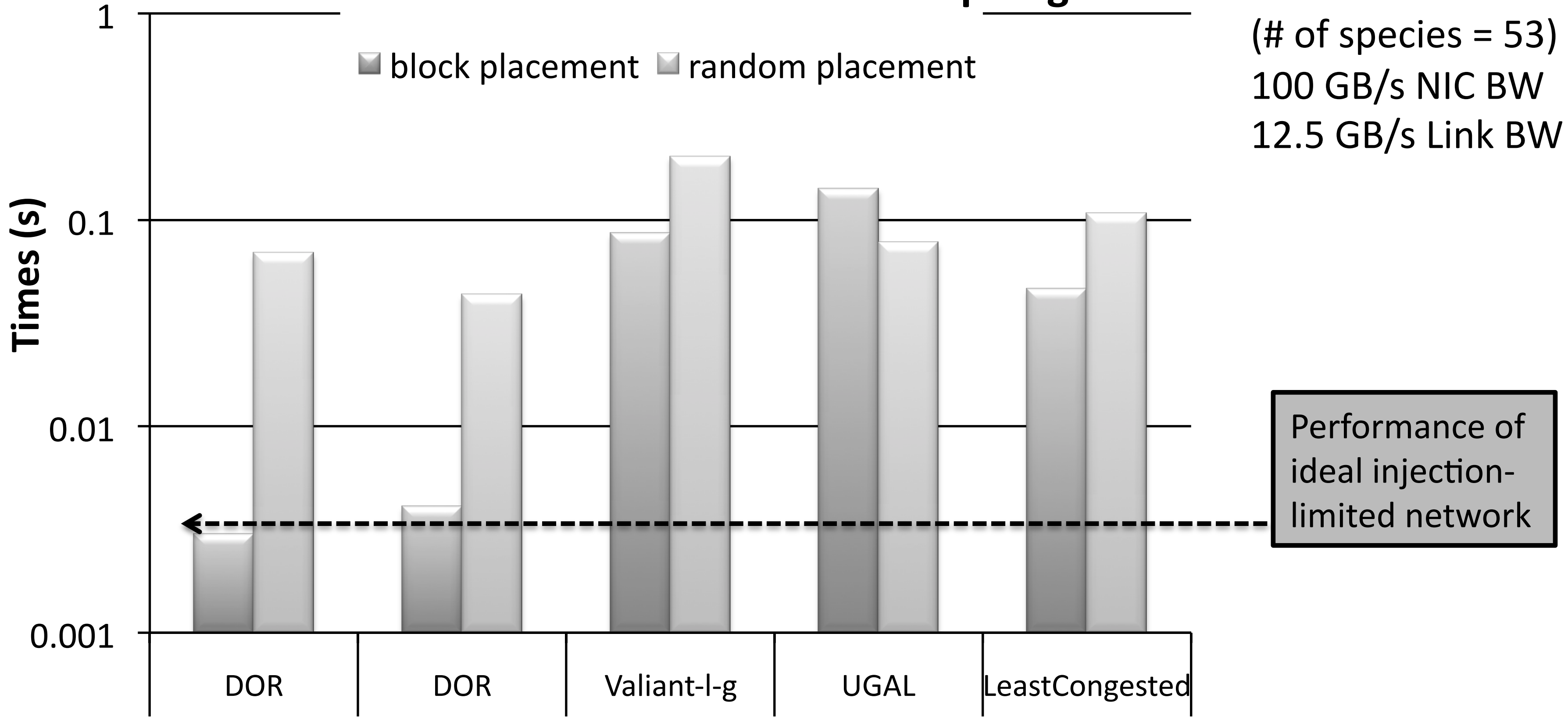


Estimated Performance Improvements



Neither software optimizations alone nor hardware optimizations alone will not get us to the exascale, we have to apply both.

16K Network End Points Communication Times on Different Topologies



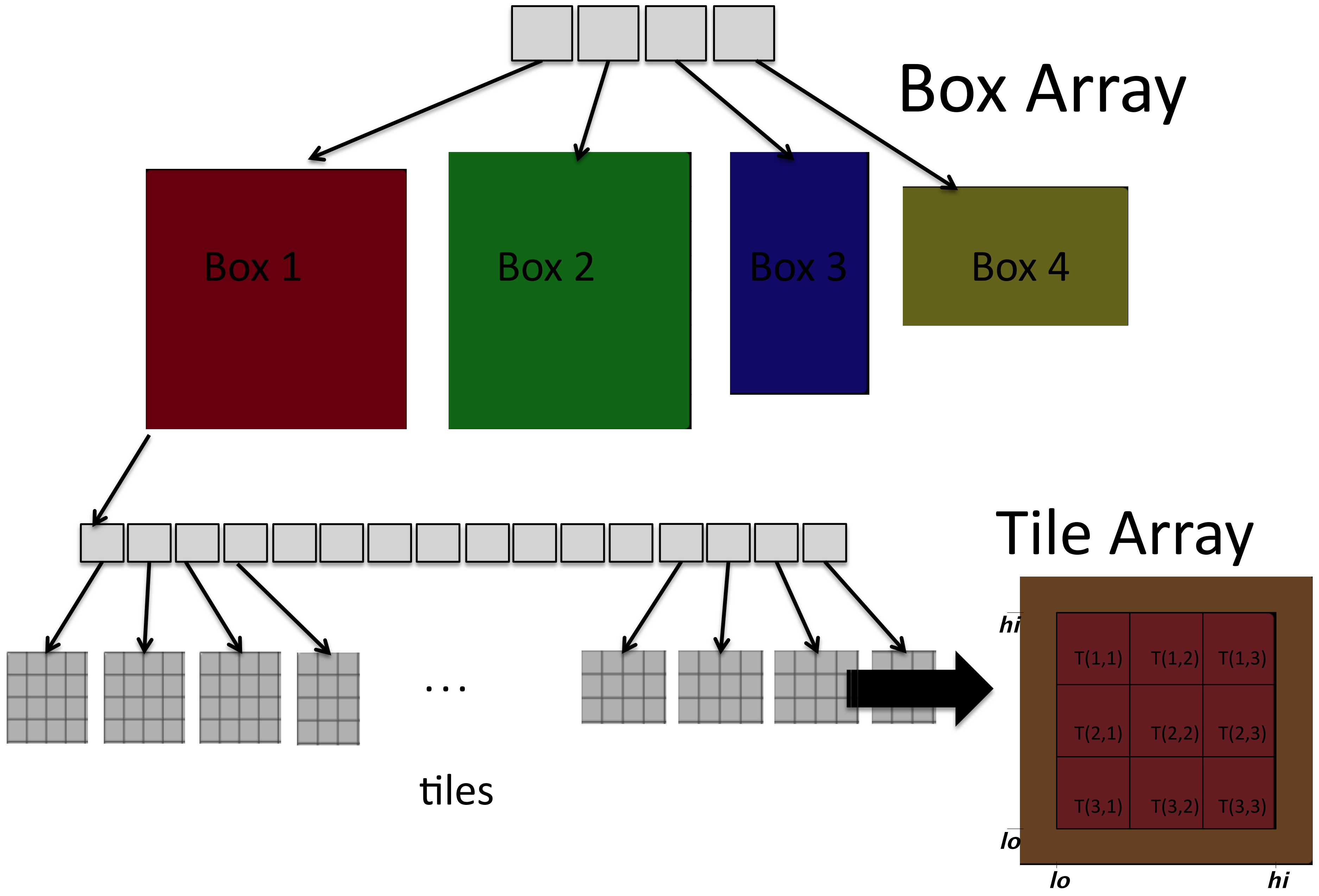
- Analytical model assumes the ideal network (dashed lines), we used SST/macro simulator to observe the performance impact of network topology
- Torus-like topologies are better-suited to combustion codes
- Job placement improves performance if topology-aware scheduling is used
  - Block: sequential numbering of ranks on sequential nodes
  - Random mixes up the ranks

Programming Model Design

- Leverage the lessons learned from ExaSAT in programming model design in context of combustion
  - Lightweight performance model identifies valuable software optimizations (and their hardware requirements) for compiler and adaptive runtime
  - Helps find optimal tuning parameters (e.g. blocking factor)
- Offer two modes of parallelism and cover all cases covered by SPMD but improve analyzability
  - Data parallel: Focus on expression of hierarchy and topology of data through tiling for locality and data movement
  - Task parallel: Focus on use of functional semantics for each task, enables asynchronous pipeline parallelism
- Embed data parallel unit within a task container
  - For example in AMR one task container per “box”, and then within that have a data parallel threads to parallelize operation on each box

Data Layout

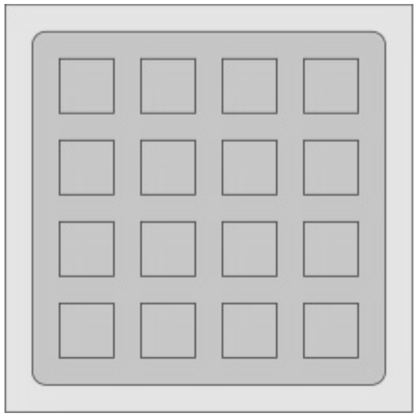
- Adopt a data-centric model
  - Describe how the data is laid out on the system and apply the computation to the data where it resides
- Use these language constructs to transfer information from the programmer to compiler and runtime
- Tiling can be expressed in the data structure
  - For example: HTA, HDFS5
- A tile represent an independent unit of work, which becomes a task, more coarse grain than single iteration

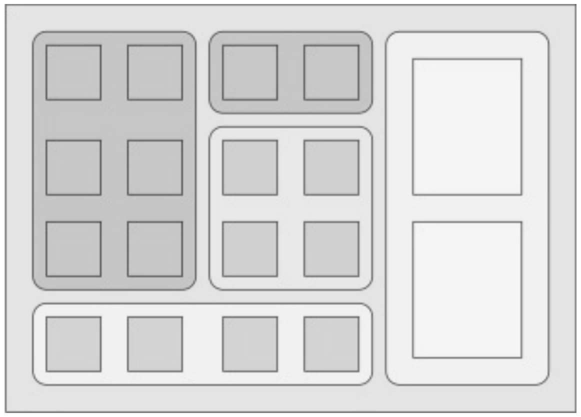

Box Array
Box 1
Box 2
Box 3
Box 4
Tile Array
...
tiles
hi
lo
T(1,1)
T(1,2)
T(1,3)
T(2,1)
T(2,2)
T(2,3)
T(3,1)
T(3,2)
T(3,3)
lo
hi

Intelligent Adaptive Runtime

- Conduct dataflow analysis of program
  - Dynamically map tasks and data to locations to improve load balance and while minimizing data movement
  - Co-locate tasks of different types to increase concurrency and minimize contention of shared resources (memory bandwidth, cache footprint, ALUs)
- Tune aggresiveness of tiling and fusion optimizations
  - Can choose parameters based on environment (e.g. available shared L3 cache)
- Automate movement of data between disjoint address spaces (e.g. local stores)

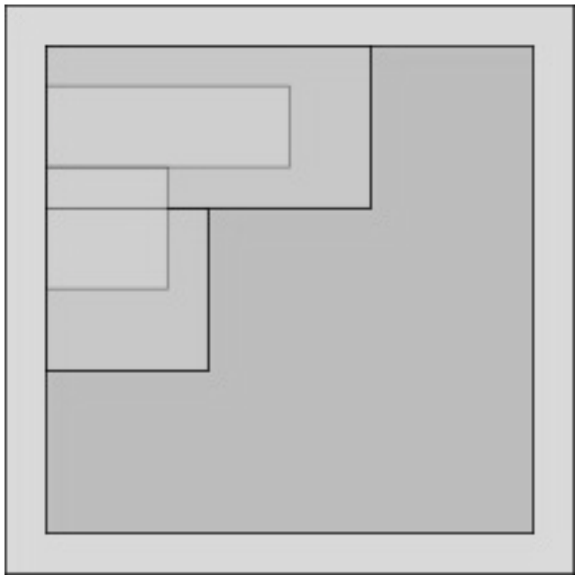
Multiple Tasks per Location

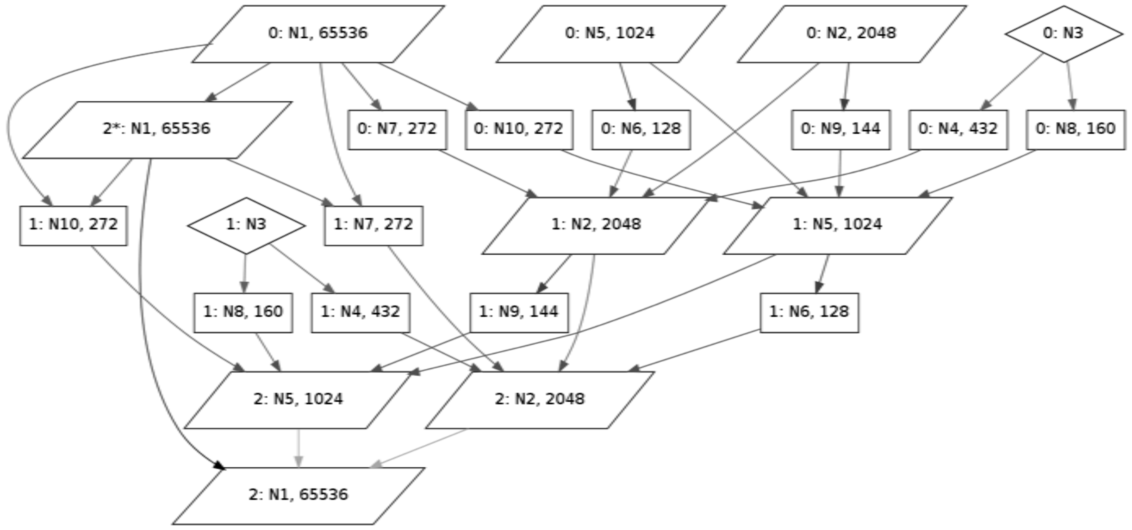

Single-task mapping


Multi-task Mapping

- OpenMP parallelizes each task over a whole processor
- Map multiple tasks to different sized subsets of cores in a single processor
  - Scheduler can be aware of both topology and heterogeneity
- Automate this process using static analysis

AMR Box Dependency and Communications Analysis





- From list of boxes, determine data dependencies and communications requirements for AMR code
- Use box index set operations (e.g. intersect, set difference) to determine required data exchange
- Can experiment with different data distributions
- Collaborating with SST/Macro group to simulate communications